Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds

Maciej Malawski*‡, Gideon Juve†, Ewa Deelman† and Jarek Nabrzyski‡
*AGH University of Science and Technology, Dept. of Computer Science, Krakow, Poland, Email: malawski@agh.edu.pl
†USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, USA, Email: {gideon,deelman}@isi.edu
‡University of Notre Dame, Center for Research Computing, 111 ITC, Notre Dame, IN, USA, Email: naber@nd.edu

***Abstract*—Large-scale applications expressed as scientific workflows are often grouped into ensembles of inter-related workflows. In this paper, we address a new and important problem concerning the efficient management of such ensembles under budget and deadline constraints on Infrastructure- as-a-Service (IaaS) clouds. We discuss, develop, and assess algorithms based on static and dynamic strategies for both task scheduling and resource provisioning. We perform the evaluation via simulation using a set of scientific workflow ensembles with a broad range of budget and deadline parameters, taking into account uncertainties in task runtime estimations, provisioning delays, and failures. We find that the key factor determining the performance of an algorithm is its ability to decide which workflows in an ensemble to admit or reject for execution. Our results show that an admission procedure based on workflow structure and estimates of task runtimes can significantly improve the quality of solutions.**

I. Introduction

Scientific workflows, usually represented as Directed Acyclic Graphs (DAGs), are an important class of applications that lead to challenging problems in resource management on grid and utility computing systems. Workflows for large computational problems are often composed of several inter-related workflows grouped into *ensembles*. Workflows in an ensemble typically have a similar structure, but they differ in their input data, number of tasks, and individual task sizes.

There are many applications that require scientific workflow ensembles. CyberShake [1], for example, uses ensembles to generate seismic hazard maps. Each workflow in a CyberShake ensemble generates a hazard curve for a particular geographic location, and several hazard curves are combined to create a hazard map. In 2009 CyberShake was used to generate a map that required an ensemble of 239 workflows. Similarly, users of Montage [2] often need several workflows with different parameters to generate a set of image mosaics that can be combined into a single, large mosaic. The Galactic Plane ensemble, which generates several mosaics of the entire sky in different wavelengths, consists of 17 workflows, each of which contains 900 sub-workflows. Another ensemble example is the Periodograms application [3], which searches for extrasolar planets by detecting periodic dips in the light intensity of their host star. Due to the large scale of the input data, this application is often split up into multiple batches processed by different workflows. Additional workflows are created to run the analysis using different parameters. A recent analysis of Kepler satellite data required three ensembles of 15 workflows.

Workflows in an ensemble may differ not only in their parameters, but also in their priority. For example, in CyberShake some geographic locations may be in heavily populated areas or in strategic locations such as power plants, while others may be less important. Scientists typically prioritize the workflows in such an ensemble so that important workflows are finished first. This enables them to see critical results early, and helps them to choose the most important workflows when the time and financial resources available for computing are limited.

Infrastructure-as-a-Service (IaaS) clouds offer the ability to provision resources on-demand according to a pay-per-use model. These systems are regarded by the scientific community as a potentially attractive source of low-cost computing resources [4], [5]. In contrast to clusters and grids, which typically offer best-effort quality of service, clouds give more flexibility in creating a controlled and managed computing environment. Clouds provide the ability to adjust resource capacity according to the changing demands of the application, often called auto-scaling. However, giving users more control also requires the development of new methods for task scheduling and resource provisioning. Resource management decisions required in cloud scenarios not only have to take into account performance-related metrics such as workflow makespan or resource utilization, but must also consider budget constraints, since the resources from commercial clouds usually have monetary costs associated with them [6].

In this paper, we aim to gain insight into resource management challenges when executing scientific workflow ensembles on clouds. We address a new and important problem of maximizing the number of completed workflows from an ensemble under both budget and deadline constraints. The motivation for this work is to answer the fundamental question of concern to a researcher: How much computation can be completed given the limited budget and timeframe of a research project?

The main contributions of this paper are:

- we define the problem of scheduling prioritized workflow ensembles under budget and deadline constraints,
- we analyze and develop several dynamic (online) and static (offline) algorithms for task scheduling and resource provisioning that rely on workflow structure information (critical paths and workflow levels) and estimates of task runtimes,
- we evaluate these algorithms using a simulator based on CloudSim [7], which models the infrastructure and

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

the application,taking into account uncertainties in task runtime estimates, provisioning delays, and failures,
- we discuss the performance of the algorithms on a set of synthetic workflow ensembles based on important, real scientific applications, using a broad range of different application scenarios and varying constraint values.

II. Related Work

General policy and rule-based approaches to dynamic provisioning (e.g. Amazon Auto Scaling [8] and RightScale [9]) allow the size of a resource pool to be adjusted based on infrastructure and application metrics. A typical infrastructure-specific metric is system load, whereas application-specific metrics include response time and length of a task or of a request queue. It is possible to set thresholds and limits to tune the behavior of these autoscaling systems, but no support for complex applications is provided. Policy-based approaches for scientific workloads (e.g. [10], [11]) also allow to scale the cloud resource pool or to extend the capabilities of clusters using cloud-burst techniques. Our approach is different in that we consider workflows, while policy based approaches typically consider bags of independent tasks or unpredictable batch workloads. This enables us to take advantage of workflow-aware heuristics that cannot be applied to independent tasks.

Our work is related to the strategies for deadline-constrained cost-minimization workflow scheduling, developed for utility grid systems. However, our problem is different from [12] and [13] in that we consider ensembles of workflows in IaaS clouds, which allow one to provision resources on a per-hour billing model, rather than utility grids, which allow one to choose from a pool of existing resources with a per-job billing model. Our work is also different from cloud-targeted autoscaling solution [14] in that we consider ensembles of workflows rather than unpredictable workloads containing workflows. We also consider budget constraints rather than cost minimization as a goal. In other words, we assume that there is more work to be done than the available budget, so some work must be rejected. Therefore, cost is not something we optimize (i.e. an objective), but rather a constraint.

This work is related to bi-criteria scheduling and multi-criteria scheduling of workflows [15], [16], [17]. These approaches are similar to ours in that we have two scheduling criteria: cost and makespan. The challenge in multi-criteria scheduling is to derive an objective function that takes into account all of the criteria. In our case one objective (amount of work completed) is subject to optimization, whereas time and cost are treated as constraints. Other approaches [18], [19] use metaheuristics that usually run for a long time before producing good results, which makes them less useful in the scenarios we consider in this paper. Our work can also be regarded as an extension of the budget-constrained workflow scheduling [20] in the sense that we are dealing with workflow ensembles and the deadline constraint is added.

III. Problem Description

Resource Model We assume a resource model similar to Amazon's Elastic Compute Cloud (EC2), where virtual machine (VM) instances may be provisioned on-demand and are billed by the hour, with partial hours being rounded up. Although there may be heterogeneous VM types with different amounts of CPU, memory, disk space, and I/O, for this paper we focus on a single VM type because we assume that for most applications there will typically be only one or two VM types with the best price/performance ratio for the application [21]. We assume that a submitted task has exclusive access to a VM instance and that there is no preemption. We also assume that there is a delay between the time that a new VM instance is requested and when it becomes available to execute tasks.

Application Model The target applications are ensembles of scientific workflows that can be modeled as DAGs, where the nodes in the graph represent computational tasks, and the edges represent data- or control-flow dependencies between the tasks. We assume that runtime estimates for the workflow tasks are known, but that they are not perfect and may vary based on a uniform distribution of $\pm p\%$.

This study uses synthetic workflows that were generated using historical data from real applications [22]. The applications come from a wide variety of domains including: bioinformatics (Epigenomics, SIPHT: sRNA identification protocol using high-throughput technology), astronomy (Montage), earthquake science (CyberShake), and physics (LIGO). The synthetic workflows were generated using code developed in [23], with task runtimes based on distributions gathered from running real workflows.

Although workflows are often data-intensive, the algorithms described here do not currently consider the size of input and output data when scheduling tasks. Instead, it is assumed that all workflow data is stored in a shared cloud storage system, such as Amazon S3, and that intermediate data transfer times are included in task runtimes. It is also assumed that data transfer times between the shared storage and the VMs are equal for different VMs so that task placement decisions do not impact the runtime of the tasks.

We assume that each workflow in an ensemble is given a numeric priority that indicates how important the workflow is to the user. As such, the priorities indicate the utility function of the user. These priorities are absolute in the sense that completing a workflow with a given priority is more valuable than completing all other workflows in the ensemble with lower priorities combined. The goal of the workflow ensemble scheduling and cloud provisioning problem is to complete as many high-priority workflows as possible given a fixed budget and deadline. Only workflows for which all tasks are finished by the deadline are considered to be complete—partial results are not usable in this model.

Performance Metric In order to precisely define the objective of the algrithms it is necessary to introduce a metric that can be used to score the performance of the different algorithms on a given problem (ensemble, budget, and deadline).

Algorithm 1 Dynamic provisioning algorithm for DPDS

Require: c: consumed budget; b: total budget; d: deadline; p: price; t: current time; u_h: upper utilization threshold; u_l: lower utilization threshold; v_{max}: maximum number of VMs

1: **procedure** PROVISION
2: $V_R \leftarrow$ set of running VMs
3: $V_C \leftarrow$ set of VMs completing billing cycle
4: $V_T \leftarrow \emptyset$ ▷ set of VMs to terminate
5: $n_T \leftarrow 0$ ▷ number of VMs to terminate
6: **if** $b - c < |V_C| * p$ *or* $t > d$ **then**
7: $n_T \leftarrow |V_R| - \lfloor (b-c)/p \rfloor$
8: $V_T \leftarrow$ select n_T VMs to terminate from V_C
9: TERMINATE(V_T)
10: **else**
11: $u \leftarrow$ current VM utilization
12: **if** $u > u_h$ and $|V_R| < v_{max} * N_{VM}$ **then**
13: START($new\ VM$)
14: **else if** $u < u_l$ **then**
15: $V_I \leftarrow$ set of idle VMs
16: $n_T \leftarrow \lceil |V_I|/2 \rceil$
17: $V_T \leftarrow$ select n_T VMs to terminate from V_I
18: TERMINATE(V_T)
19: **end if**
20: **end if**
21: **end procedure**

The simplest approach is to count the number of workflows in the ensemble that each algorithm is able to complete within the budget before the deadline, but this metric does not account for the priority-based utility function specified by the user. Using the counting approach, a less efficient algorithm may be able to complete a large number of low-priority workflows by executing the smallest workflows first. In order to account for the priority, we use an exponential scoring defined as:

$$Score(e) = \sum_{w \in Completed(e)} 2^{-Priority(w)}$$

where $Completed(e)$ is the set of workflows in ensemble e that was completed by the algorithm, and $Priority(w)$ is the priority of workflow w such that the highest-priority workflow has $Priority(w) = 0$, the next highest workflow has $Priority(w) = 1$, and so on. This exponential scoring function gives the highest priority workflow a score that is higher than all the lower- priority workflows combined:

$$2^{-p} > \sum_{i = p+1,\ ...} 2^{-i}$$

IV. ALGORITHMS

This section describes three algorithms that were developed to schedule and provision resources for ensembles of workflows on the cloud under budget and deadline constraints.

A. Dynamic Provisioning Dynamic Scheduling (DPDS)

DPDS is an online algorithm that provisions resources and schedules tasks at runtime. It consists of two main parts: a provisioning procedure, and a scheduling procedure.

DPDS' provisioning procedure is based on resource utilization. DPDS starts with a fixed number of resources calculated based on the available time and budget, and adjusts the number of resources according to how well they are utilized by the application. Given a budget in dollars b, deadline in hours d, and the hourly price of a VM in dollars p, it is possible to calculate the number of VMs, N_{VM}, to provision so that the entire budget is consumed before the deadline:

$$N_{VM} = \lceil b/(d * p) \rceil \quad (1)$$

DPDS provisions N_{VM} VMs at the start of the ensemble execution, then it periodically computes resource utilization using the percentage of idle VMs over time and adjusts the number of VMs if the utilization is above or below given thresholds. Because it is assumed that VMs are billed by the hour, DPDS only considers VMs that are approaching their hourly billing cycle when deciding which VMs to terminate. This dynamic provisioning algorithm is shown in Algorithm 1.

The set of VMs completing their billing cycle is determined by both the provisioner interval, and the termination delay of the provider. This guarantees that VMs can be terminated before they start the next billing cycle and prevents the budget from being overrun. The VMs terminated in line 9 of Algorithm 1 are the ones that would overrun the budget if not terminated in the current provisioning cycle. The VMs terminated in line 18 are chosen to increase the resource utilization to the desired threshold. In order to prevent instances that have already been paid for from being terminated too quickly, no more than half of the idle resources are terminated during each provisioning cycle. To avoid an uncontrolled increase in the number of instances, which may happen in the case of highly parallel workflows, the provisioner will not start a new VM if the number of running VMs is greater than the product of N_{VM} (from Equation 1) and an autoscaling parameter, v_{max}. Unless otherwise specified, v_{max} is assumed to be 1.

In order to schedule individual workflow tasks onto available VMs, DPDS uses the dynamic, priority-based scheduling procedure shown in Algorithm 2. Initially, the ready tasks from all workflows in the ensemble are added to a priority queue based on the priority of the workflow to which they belong. If there are idle VMs available, and the priority queue is not empty, the next task from the priority queue is submitted to an arbitrarily chosen idle VM. The process is repeated until there are no idle VMs or the priority queue is empty. The scheduler

Algorithm 2 Priority-based scheduling algorithm for DPDS

1: **procedure** SCHEDULE
2: $P \leftarrow$ empty priority queue
3: $IdleVMs \leftarrow$ set of idle VMs
4: **for** root task t in all workflows **do**
5: INSERT(t, P)
6: **end for**
7: **while** deadline not reached **do**
8: **while** $IdleVMs \neq \emptyset$ *and* $P \neq \emptyset$ **do**
9: $v \leftarrow$ SELECTRANDOM($IdleVMs$)
10: $t \leftarrow$ POP(P)
11: SUBMIT(t, v)
12: **end while**
13: Wait for task t to finish on VM v
14: Update P with ready children of t
15: INSERT($v, IdleVMs$)
16: **end while**
17: **end procedure**

Algorithm 3 Workflow admission algorithm for WA-DPDS

Require: w: workflow; b: budget; c: current cost
1: **procedure** ADMIT(w, b, c)
2: $r_n \leftarrow b - c$ ▷ Budget remaining for new VMs
3: $r_c \leftarrow$cost committed to VMs that are running
4: $r_a \leftarrow$cost to complete workflows previously admitted
5: $r_m \leftarrow 0.1$ ▷ Safety margin
6: $r_b \leftarrow r_n + r_c - r_a - r_m$ ▷ Budget remaining
7: $c_w \leftarrow$ ESTIMATECOST(w)
8: **if** $c_w < r_b$ **then return** $TRUE$
9: **else return** $FALSE$
10: **end if**
11: **end procedure**

Algorithm 4 Ensemble planning algorithm for SPSS

Require: W: workflow ensemble; b: budget; d: deadline
Ensure: Schedule as much of W as possible given b and d
1: **procedure** PLANENSEMBLE(W, b, d)
2: $P \leftarrow \emptyset$ ▷ Current plan
3: $A \leftarrow \emptyset$ ▷ Set of admitted DAGs
4: **for** w *in* W **do**
5: $P' \leftarrow$ PLANWORKFLOW(w, P, d)
6: **if** $Cost(P') \leq b$ **then**
7: $P \leftarrow P'$ ▷ Accept new plan
8: $A \leftarrow A + w$ ▷ Admit w
9: **end if**
10: **end for**
11: **return** P, A
12: **end procedure**

then waits for a task to finish, adds its ready children to the priority queue, marks the VM as idle, and the entire process repeats until the deadline is reached.

DPDS guarantees that tasks from lower priority workflows are always deferred when higher-priority tasks are available, but lower-priority tasks can still occupy idle VMs when higher-priority tasks are not available. Since there is no preemption, long-running low-priority tasks may delay the execution of higher-priority tasks. In addition, tasks from low priority workflows may be executed even though there is no chance that those workflows will be completed within the current budget and deadline. Fig. 1a shows an example schedule generated using the DPDS algorithm. The figure illustrates how tasks from lower priority workflows backfill idle VMs when tasks from higher priority workflows are not available.

B. Workflow-Aware DPDS (WA-DPDS)

DPDS does not use any information about the structure of the workflows in the ensemble when scheduling tasks. It does not consider whether a lower priority task belongs to a workflow that will never be able to complete given the current budget and deadline. As a result, DPDS may start lower priority tasks just to keep VMs busy that will end up delaying higher priority tasks later on, making it less likely that higher priority workflows will be able to finish.

The Workflow-Aware DPDS (WA-DPDS) algorithm extends DPDS by introducing a workflow admission procedure, which is invoked whenever WA-DPDS sees the first task of a new workflow at the head of the priority queue (i.e. when no other tasks from the workflow have been scheduled yet). The admission procedure— shown in Algorithm 3—estimates whether there is enough budget remaining to admit the new workflow; if there is not, then the workflow is rejected and its tasks are removed from the queue. WA-DPDS compares the current cost (consumed budget) and remaining budget, taking into account the cost of currently running VMs, and the cost of workflows that have already been admitted. In addition, it adds a small safety margin of \$0.10 (10% of the compute hour cost) to avoid going over budget. We found the admission procedure useful not only to prevent low-priority workflows from delaying high-priority ones, but also to reject large and costly workflows that would overrun the budget and admit smaller workflows that can efficiently utilize idle resources.

C. Static Provisioning Static Scheduling (SPSS)

The previous dynamic (online) algorithms make provisioning and scheduling decisions at runtime. By contrast, the SPSS algorithm creates a provisioning and scheduling plan before running any workflow tasks. This enables SPSS to start only those workflows that it knows can be completed given the deadline and budget constraints, and eliminates any waste that may be allowed by the dynamic algorithms. However, the disadvantage of SPSS is that it is sensitive to dynamic changes in the environment (see Sections VI-C and VI-B).

Algorithm 4 shows how ensembles are planned in SPSS. Workflows from the ensemble are considered in priority order. For each workflow, SPSS attempts to build on top of the current plan by provisioning VMs to schedule the tasks of the workflow so that it finishes before the deadline with the least possible cost. If the cost of the new plan is less than the budget, then the new plan is accepted and the workflow is admitted. If not, then the new plan is rejected and the process continues with the next workflow in the ensemble. The idea is that, if each workflow can be completed by the deadline with the lowest possible cost, then the number of workflows that can be completed within the given budget will be maximized.

To plan a workflow, the SPSS algorithm assigns sub-deadlines to each individual task in the workflow, and then schedules each task so as to minimize the cost of the task while still meeting its assigned sub-deadline. If each task can be completed by its deadline in the least expensive way, then the cost of the entire workflow can be minimized without exceeding the deadline. SPSS assigns sub-deadlines to each task based on the slack time of the workflow, which is defined as the amount of extra time that a workflow can extend its critical path and still be completed by the ensemble deadline. For a workflow w, the slack time of w is: $ST(w) = d - CP(w)$ where d is the deadline and $CP(w)$ is the critical path of w. We assume that $CP(w) \leq d$, otherwise the workflow cannot be completed by the deadline and must be rejected.

A task's level is the length of the longest path between the task and an entry task of the workflow:

$$Level(t) = \begin{cases} 0, & \text{if } Pred(t) = \emptyset \\ \max_{p \in Pred(t)} Level(p) + 1, & \text{otherwise.} \end{cases}$$

Algorithm 5 Workflow planning algorithm for SPSS

Require: w: workflow; P: current plan; d: deadline
Ensure: Create plan for w that minimizes cost and meets deadline d

```
1: procedure PLANWORKFLOW(w, P, d)
2:     P' ← copy of P
3:     DEADLINEDISTRIBUTION(w,d)
4:     for t in w sorted by DL(t) do
5:         v ← VM that minimizes cost and start time of t
6:         if FinishTime(t, v) < DL(t) then
7:             Schedule(t,v)
8:         else
9:             Provision a new VM v
10:            Schedule(t,v)
11:        end if
12:    end for
13:    return P'
14: end procedure
```

SPSS distributes the slack time of the workflow by level, so that each level of the workflow gets a portion of the workflow's slack time proportional to the number of tasks in the level and the total runtime of tasks in the level. The idea is that levels containing many tasks and large runtimes should be given a larger portion of the slack time so that tasks in those levels may be serialized. Otherwise, many resources need to be allocated to run all of the tasks in parallel, which may be more costly.

The slack time of a level l in workflow w is given by:

$$ST(l) = ST(w)\left[\left(\alpha\frac{N(l)}{N(w)}\right) + \left((1-\alpha)\frac{R(l)}{R(w)}\right)\right]$$

where $N(w)$ is the number of tasks in the workflow, $N(l)$ is the number of tasks in level l, $R(w)$ is the total runtime of all tasks in the workflow, $R(l)$ is the total runtime of all tasks in level l, and α is a parameter between 0 and 1 that causes more slack time to be given to levels with more tasks (large α) or more runtime (small α).

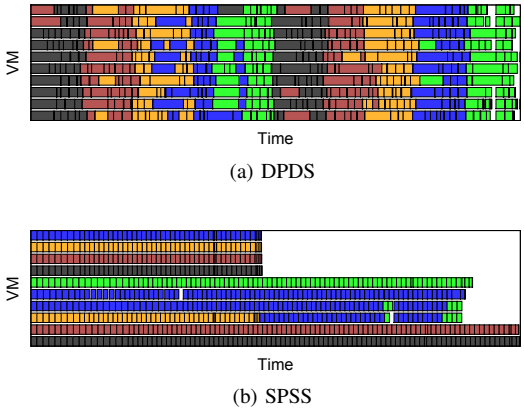
The deadline of a task t is then:

$$DL(t) = LST(t) + RT(t) + ST(Level(t))$$

where $Level(t)$ is the level of t, $RT(t)$ is the runtime of t, and $LST(t)$ is the latest start time of t determined by:

$$LST(t) = \begin{cases} 0, & \text{if } Pred(t) = \emptyset \\ \max_{p \in Pred(t)} DL(p), & \text{otherwise.} \end{cases}$$

Algorithm 5 shows how SPSS creates low-cost plans for each workflow. The PLANWORKFLOW procedure first calls DEADLINEDISTRIBUTION to assign sub-deadlines to tasks. Then, PLANWORKFLOW schedules tasks onto VMs, allocating new VMs when necessary. For each task in the workflow, the least expensive slot is chosen to schedule the task so that it can be completed by its deadline. VMs are allocated in blocks of one billing cycle (one hour) regardless of the size of the task. When computing the cost of scheduling a task on a given VM, the algorithm considers idle slots in blocks that were allocated for previous tasks to be free, while slots in new blocks cost the full price of a billing cycle. For example, if a task has a runtime of 10 minutes, and the price of a block is $1, then the algorithm will either schedule the task on an existing VM that has an idle slot larger than 10 minutes for a cost of $0, or it will allocate a new block on an existing VM, or provision a new VM, for a cost of $1. If the cost of slots on two different VMs is equal, then the slot with the earliest start time is chosen. To prevent too many VMs from being provisioned, the algorithm always prefers to extend the runtime of existing VMs before allocating new VMs.



(a) DPDS

(b) SPSS

Fig. 1: Example schedules generated by the algorithms. Each row is a different VM. Boxes are tasks colored by workflow.

An example schedule generated by SPSS (Fig. 1b) shows how SPSS tends to start many workflows in parallel, running each workflow over a longer period of time on only a few VMs to minimize cost. In comparison, the dynamic algorithms tend to run one workflow at a time across many VMs in parallel.

V. EVALUATION METHODS

A. Simulator

To evaluate and compare the three proposed algorithms, we developed a cloud workflow simulator [24] based on CloudSim [7]. Our simulation model consists of Cloud, VM and WorkflowEngine entities. The Cloud entity starts and terminates VM entities using an Amazon EC2-like API. VM entities simulate the execution of individual tasks, including randomized variations in runtime. The WorkflowEngine entity manages the scheduling of tasks and the provisioning of VMs based on the chosen algorithm. We assume that the VMs have a single core and execute tasks sequentially. The simulator reads workflow description files in a modified version of the DAX format used by the Pegasus Workflow Management System [25].

B. Workflow Ensembles

In order to evaluate the algorithms on a standard set of workflows, we created randomized ensembles using workflows available from the workflow generator gallery [23]. The gallery contains synthetic workflows modeled using structures and parameters that were taken from real applications. Ensembles were created using synthetic workflows from five real applications: SIPHT, LIGO, Epigenomics, Montage and CyberShake. For each application, workflows with 50, 100, 200, 300, 400,

500, 600, 700, 800, 900 and 1000 tasks were created. For each workflow size, 20 different workflow instances were generated using parameters and task runtime distributions from real workflow traces. The total collection of synthetic workflows contains 5 applications, 11 different workflow sizes, and 20 workflow instances, for a total of 1100 synthetic workflows.

Using this collection of workflows, we constructed five different ensemble types: constant, uniform sorted, uniform unsorted, Pareto sorted and Pareto unsorted. In the unsorted ensembles, workflows of different sizes are mixed together and the priorities are assigned randomly. For many applications, however, large workflows are more important to users than small workflows because they represent more significant computations. To model this, the sorted ensembles are sorted by size, so that the largest workflows have the highest priority.

Constant ensembles contain workflows that all have the same number of tasks. The number of tasks is chosen randomly from the set of possible workflow sizes. Once the size is determined, then N workflows of that size are chosen randomly for the ensemble from the set of synthetic workflows.

Uniform ensembles contain workflows with sizes that are uniformly distributed among the set of possible sizes. Each workflow is selected by first randomly choosing the size of the workflow and then randomly choosing a workflow of that size from the set of synthetic workflows.

Pareto ensembles contain a small number of larger workflows and a large number of smaller workflows. Their sizes are chosen according to a Pareto distribution. The distribution was modified so that the number of large workflows (of size $\geq$ 900) is increased by a small amount to produce a "heavy-tail". This causes Pareto ensembles to have a slightly larger number of large workflows, which reflects behavior commonly observed in many computational workloads.

The number of workflows in an ensemble depends on the particular application, but we assume that ensemble sizes are on the order of between 10 and 100 workflows, typical of the real applications we have examined (see Section I).

C. Experimental Parameters

In order to observe the interesting characteristics of the proposed algorithms, for each ensemble, we selected ranges for deadline and budget that cover a broad parameter space: from tight constraints, where only a small number of workflows can be completed, to more liberal constraints where all, or almost all, of the workflows can be completed. We computed constraint ranges based on the characteristics of each ensemble. The budget constraints are calculated by identifying the smallest budget required to execute one of the workflows in the ensemble (MinBudget), and the smallest budget required to execute all workflows in the ensemble (MaxBudget):

$$MinBudget = \min_{w \in e} Cost(w)$$

$$MaxBudget = \sum_{w \in e} Cost(w)$$

This range—$[MinBudget, MaxBudget]$—is then divided into equal intervals to determine the budgets to use in each experiment. Similarly, the deadline constraints are calculated by identifying the smallest amount of time required to execute a single workflow in the ensemble (MinDeadline), which is the length of the critical path for the workflow with the shortest critical path, and by identifying the smallest amount of time required to execute all workflows (MaxDeadline), which is the sum of the critical paths of all the workflows:

$$MinDeadline = \min_{w \in e} CriticalPath(w)$$

$$MaxDeadline = \sum_{w \in e} CriticalPath(w)$$

This range—$[MinDeadline, MaxDeadline]$—is then divided into equal intervals. By computing the budget and deadline constraints in this way we ensure that the experiments for each ensemble cover the most interesting area of the parameter space for the ensemble.

In all the experiments we assumed that the VMs have a price of $1 per VM-hour. This price was chosen to simplify interpretation of results and should not affect the relative performance of the different algorithms. In this study the heterogeneity of the infrastructure is not relevant since we assume that it is always possible to select a VM type that has the best price to performance ratio for a given application [21].

All the experiments were run with maximum autoscaling factor (v_{max}) set to 1.0 for DPDS and WA-DPDS. After experimenting with DPDS and WA-DPDS we found that, due to the high parallelism of workflows used, the resource utilization remains high enough without adjusting the autoscaling rate. Based on experiments with the target applications, we set the SPSS α parameter for deadline distribution to be 0.7, which allocates slightly more time to levels with many tasks.

VI. Discussion of Results

A. Relative Performance of Algorithms

The goal of the first experiment is to characterize the relative performance of the proposed algorithms. This was done by simulating the algorithms on many different ensembles and comparing the scores computed using the performance metric based on exponential scoring defined in Section III.

Fig. 2 shows the percentage of simulations for which each algorithm achieved the highest score for a given ensemble type. This experiment was conducted using all five applications, with all five types of ensembles. For each application and ensemble type, 10 random ensembles of 50 workflows each were created. Each ensemble was simulated with all three algorithms using 10 budgets and 10 deadlines (1,000 simulations per application, ensemble type, and algorithm, or 75,000 in total). The best scores percentage is computed by counting the number of times that a given algorithm achieved the highest score and dividing by 1000. Note that it is possible for multiple algorithms to get the same high score (to tie), so the numbers do not necessarily add up to 100%. The sum is much higher than 100% in cases where the dynamic algorithms perform relatively well because DPDS and WA-DPDS, which are very similar algorithms, often get the same high score.

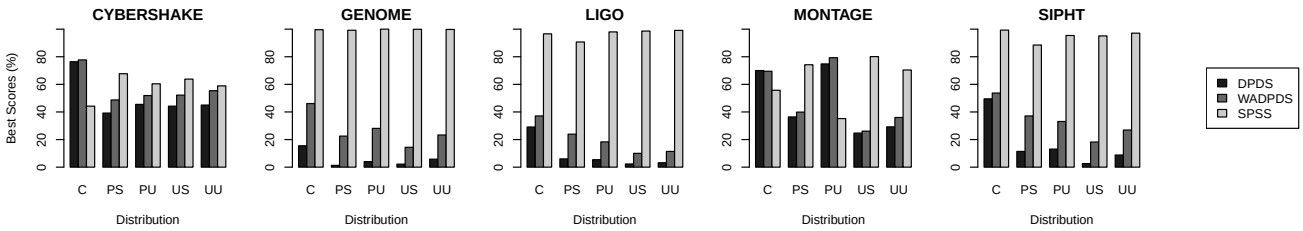


Fig. 2: Percentage of high scores achieved by each algorithm on different ensemble types for all five applications. C = Constant ensembles, PS = Pareto Sorted ensembles, PU = Pareto Unsorted ensembles, US = Uniform Sorted ensembles, and UU = Uniform Unsorted ensembles.

There are several interesting things to notice about Fig. 2. The first is that, in most cases, SPSS significantly outperforms both dynamic algorithms (DPDS and WA-DPDS). This is attributed to the fact that SPSS is able to make more intelligent scheduling and provisioning decisions because it has the opportunity to compare different options and choose the one that results in the best outcome. In comparison, the dynamic algorithms are online algorithms and are not able to project into the future to weigh the outcomes of their choices.

The second thing to notice is that, for constant ensembles, the dynamic algorithms perform significantly better relative to SPSS compared to other ensemble types. This is a result of the fact that, since all of the workflows are of approximately the same size and shape, the choice of which workflow to execute next has a smaller impact on the final result.

We can also see that the workflow-aware algorithms (WA-DPDS and SPSS) both perform better in most cases than the simple online algorithm that uses resource utilization alone to make provisioning decisions (DPDS). This suggests that there is a significant value in having information about the structure and estimated runtime of a workflow when making scheduling and provisioning decisions.

Finally, it is interesting that, for Montage and CyberShake, the relative performance of SPSS is significantly less than it is for other applications. We attribute this to the structure of Montage and CyberShake. The workflows for both applications are very wide relative to their height, and both have very short-running tasks, resulting in relatively short critical paths that makes them look more like bag-of-tasks applications, which are easier to execute than more structured applications. DPDS and WA-DPDS are able to pack more of the tasks into the available budget and deadline because there are a) more choices for where to place the tasks, and b) the different choices have a smaller impact on the algorithms' ability to execute the workflow within the constraints. In addition, the short critical paths put SPSS at a disadvantage. Because of the way SPSS assigns deadlines to individual tasks, it is prevented from starting workflows late, which prevents it from packing tasks into the idle VM slots at the end of the schedule.

B. Inaccurate Task Runtime Estimates

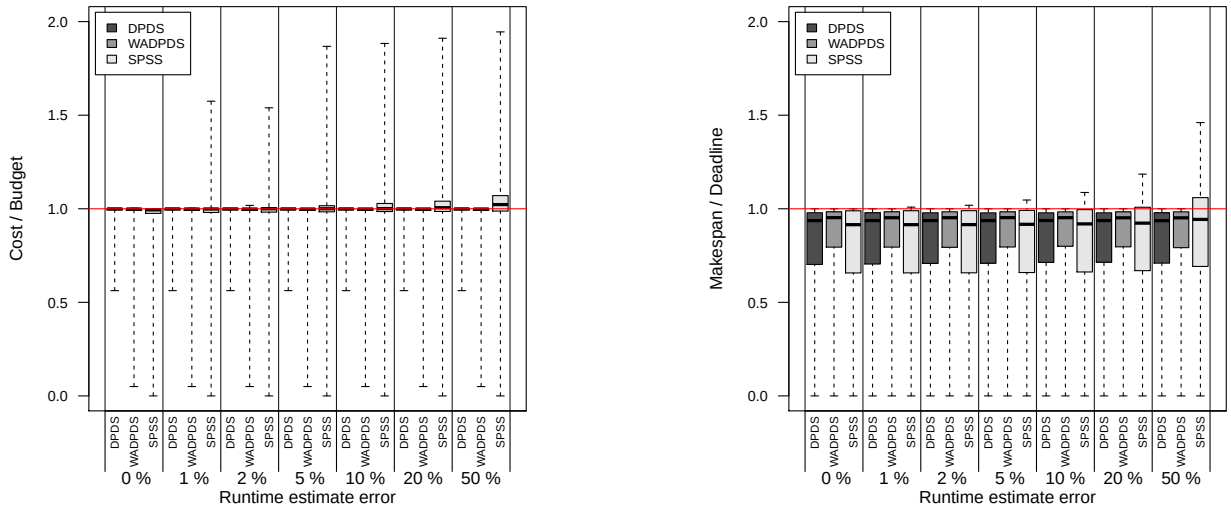
Both of the workflow-aware algorithms rely on estimates of task runtimes to make better scheduling and provisioning decisions. Our experience suggests that such assumption is often reasonable, since we can obtain workflow performance characteristics from preliminary runs [22], [25], [26]. Some applications, like Periodograms [3] even include a performance model that estimates task runtimes and automatically annotates workflow description with these data. In practice, however, these estimates are often inaccurate. Given inaccuate estimates, the question is: How do errors in task runtime estimates impact the performance of our scheduling and provisioning algorithms? To examine this we introduced uniform errors in the task runtime and observed the behavior of the algorithms in terms of meeting the desired budget and deadline constraints.

In this experiment the actual runtime of each task is adjusted in the simulation by adding a random error to the estimated runtime of $\pm p\%$. Since the sampling is done uniformly, we expect to get just as many overestimates as underestimates in any given simulation.

Fig. 3 shows the results for estimate errors ranging from 0% to 50%. The figure summarizes the outcome of an extensive suite of 525,000 simulations (10 ensembles of 50 workflows x 5 applications x 5 distributions x 10 budgets x 10 deadlines x 7 error values x 3 algorithms). Box plots show the ratio of the ensemble cost to budget, and of ensemble makespan to deadline. Whiskers on the plots indicate maximum and minimum values. The ratio indicates whether the value (for example, the simulated cost) exeeded the constraint (the budget). Values greater than 1 indicate that the constraint was exceeded.

Fig. 3a shows the ratio of simulated cost to budget. This plot illustrates two important algorithm characteristics. First, the dynamic algorithms very rarely exceeded the budget, even with very large errors. This indicates that the dynamic algorithms are able to adapt to uncertainties at runtime to ensure that the constraints are not exceeded, even when the quality of information available to them is low. Second, unlike the dynamic algorithms, the static algorithm frequently exceeded the budget constraint; by large amounts in some cases. This is a result of the fact that the static algorithm makes all of its decisions ahead of the execution and is not able to adapt to changing circumstances.

Fig. 3b shows the ratio of simulated makespan to budget. Interestingly, the deadline constraint is rarely exceeded, even in cases with very poor quality estimates. For the dynamic algorithms this is a result of the fact that they can adapt to poor estimates and stop submitting new tasks when the



(a) Ratio of Simulated Cost to Budget

(b) Ratio of Simulated Makespan to Deadline

Fig. 3: Boxplots for budget and deadline ratios when runtime estimate error varies from ±0% to ±50% for all three algorithms. Values greater than 1 indicate that the budget/deadline constraint was exceeded.

constraints are reached. Makespan is then computed as the finish time of the last fully completed workflow from the ensemble. For the static algorithm this is a result of the way that SPSS schedules workflows, and not of a particularly clever optimization. The SPSS algorithm tends to schedule workflows early, using up the budget long before the deadline is reached. This is a consequence of the deadline distribution function in SPSS, which prevents workflows from starting late. This is illustrated by the gantt chart in Fig. 1b, which shows how SPSS tends to pile up workflows at the beginning of the timeline. As a result, when the runtime of the plan is increased by introducing errors, the SPSS plan has some room to expand without exceeding the deadline.

Note that the algorithms have *not* been changed to account for inaccurate runtime estimates in this experiment. It is likely that better performance could be achieved if the algorithms were given a hint as to the accuracy of the task runtimes. Investigating that optimization is left for future work.

C. Provisioning Delays

One important issue to consider when provisioning resources in the cloud is the amount of time between when a resource is requested, and when it actually becomes available to the application. Typically these provisioning delays are on the order of a few minutes, and are highly dependent upon the cloud architecture and/or the size of the VM image [27].

We assume that resources are billed from the minute that they are requested until they are terminated. As a result, provisioning delays have an impact on both the cost and makespan of an ensemble.

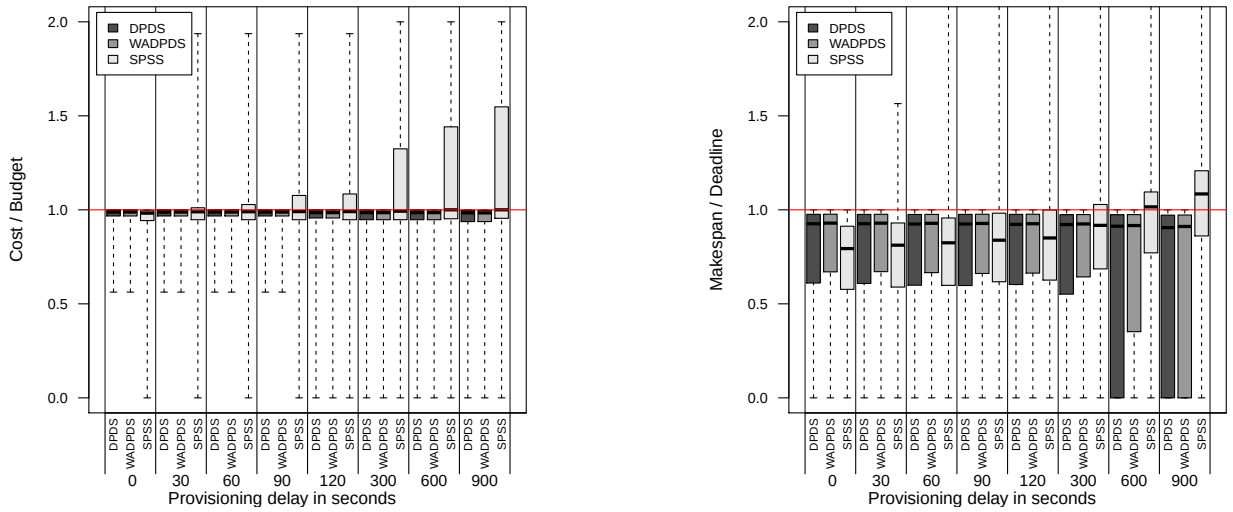
Fig. 4 shows the ratios of simulated values to constraints when the provisioning delay is increased from 0 seconds up to 15 minutes. The figure summarizes a suite of 105,000 simulations (10 ensembles of 50 workflows x 5 distributions x 10 budgets x 10 deadlines x 7 error values x 3 algorithms). To reduce the simulation time, only one application, Montage, was used in this experiment.

The effect of provisioning delays on workflow performance is similar to that of inaccurate runtime estimates: when the delays are small, all algorithms are able to produce results within the constraints, but for larger delays, the dynamic algorithms are able to adapt to avoid exceeding the constraints while the static algorithm is not. In the case of delays of more than one minute, which is typical of what has been observed on academic clouds [28] such as Magellan [29] and FutureGrid [30], approximately half of the SPSS simulations exceeded the budget; in some cases by up to a factor of 2. In comparison, none of the simulations that used a dynamic algorithm exceeded the budget or the deadline constraint.

This experiment suggests that SPSS is too sensitive to provisioning delays in its current form to be of practical use in real systems. It is possible that modifying the SPSS algorithm to account for provisioning delays would improve its performance on this experiment. In fact, since provisioning operations are infrequent (because all the algorithms tend to provision resources for a long time), it is likely that the performance of SPSS could be improved significantly by simply adding an estimate of the provisioning delay to its scheduling function. Such an estimate may not have to be particularly accurate to get good results, and developing an estimate from historical data should be relatively simple. Testing this idea is left for future work.

D. Task Failures

Running workflows consisting of large numbers of tasks on distributed systems often results in failures. The goal of the



(a) Ratio of Simulated Ensemble Cost to Budget

(b) Ratio of Simulated Ensemble Makespan to Deadline

Fig. 4: Boxplots for budget and deadline ratios when provisioning delay varies from 0 seconds to 15 minutes for all three algorithms. Values greater than 1 indicate that the budget/deadline constraint was exceeded.

next experiment was to assess the behavior of the algorithms in the presence of task execution failures by introducing a failure model into the simulator. The model is characterized by a failure rate f that defines the probability that a task will fail. The failure time of the task is determined by randomly sampling a value between the task start time and finish time. If the task fails, it is reported to the workflow engine, which retries the task until it succeeds. The dynamic algorithms re-add the task to the priority queue (queue P in Algorithm 2) so that it can be resubmitted by the scheduler. The SPSS algorithm immediately resubmits the failed task to the same VM that was selected in the plan (to minimize the disruption to the overall plan).

Fig. 5 shows the ratios of simulated values to constraints when the failures are introduced. The figure summarizes a suite of 525,000 simulations (10 ensembles of 50 workflows x 5 applications x 5 distributions x 10 budgets x 10 deadlines x 7 failure rates x 3 algorithms). As in the previous experiments, these results show that high failure rates can degrade the performance of the static algorithm considerably, while the dynamic algorithms are able to adapt. Comparing Fig. 5 to Figs. 3 and 4 one may conclude that failures are worse than provisioning delays and runtime estimate errors, since their impact is larger. However, we consider higher failure rates as rare events that suggest a significant system malfunction or invalid selection of resources.

E. SPSS Planning Time

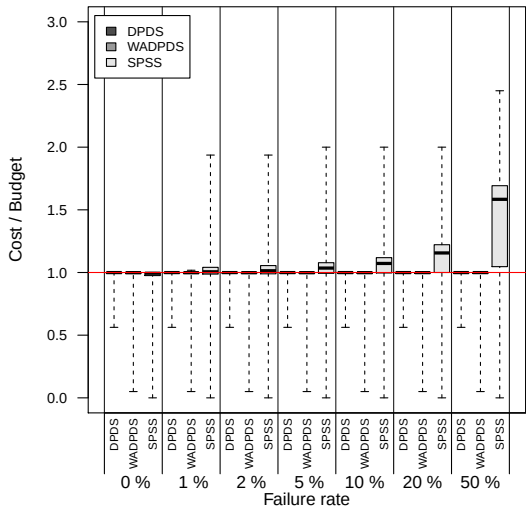
Because SPSS involves more complicated logic than the dynamic algorithms and makes its decisions before execution, it is important to understand what impact planning time has on the overall execution time.

Fig. 6 shows the SPSS planning time for ensembles of 100 workflows with five different workflow sizes: 50, 200, 400, 600, and 800 tasks. The ensembles were generated using a constant distribution equal to the workflow size desired. Two different applications were used: SIPHT and CyberShake. Each box summarizes the results of 2000 simulations (2 applications x 10 ensembles x 10 budgets x 10 deadlines).

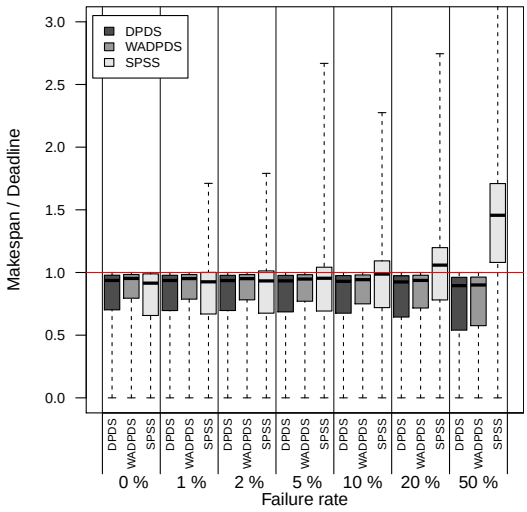
Fig. 6 shows that, for small workflows, the SPSS planning time is reasonable, taking on the order of tens of seconds to a few minutes. For larger ensembles of large workflows, however, the SPSS planning time can easily reach 10 minutes. Considering that the largest workflows used in this experiment are still relatively small (maximum of 800 tasks), and that real workflows are often much larger (workflows with tens of thousands of tasks are common, and even workflows with millions of tasks are possible), it is unlikely that SPSS will be practical for ensembles of very large workflows.

SPSS considers scheduling each task on the cheapest available slot, which involves scanning all of the available slots on all of the VMs. Since the number of available slots, in the worst case, is proportional to the number of tasks scheduled (because scheduling a task splits an existing slot into at most two slots: one before the task, and one after), the complexity of SPSS is $O(n^2)$, where n is the number of tasks in the ensemble. In comparison, the dynamic algorithms all have a more scalable complexity of $O(n)$. DPDS only examines the tasks in the workflow once when they are scheduled, and WADPDS does it twice: once in the admission algorithm, and once when they are scheduled. This makes the dynamic algorithms a better fit for larger workflows and ensembles even though in some cases they may not produce as good results as SPSS.

It may be possible to optimize SPSS to reduce its runtime by, for example, clustering the workflow to increase task granularity, which would decrease the ratio of planning time to ensemble makespan. It may also be possible to reduce the



(a) Ratio of Simulated Ensemble Cost to Budget



(b) Ratio of Simulated Ensemble Makespan to Deadline

Fig. 5: Boxplots for budget and deadline ratios when failure rate varies from 0% to 50% for all three algorithms. Values greater than 1 indicate that the budget/deadline constraint was exceeded.

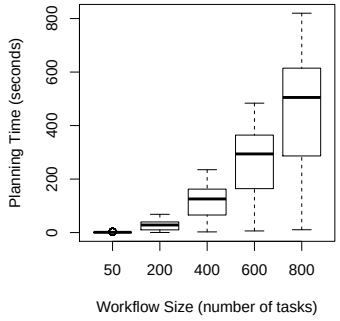


Fig. 6: Planning time of SPSS algorithm for ensembles of 100 workflows and different workflow sizes.

complexity of SPSS by employing more sophisticated data structures to store the available slots. Investigating these topics is left for future work.

VII. Conclusions and Future Work

In this paper we addressed the interesting and important new problem of scheduling and resource provisioning for scientific workflow ensembles on IaaS clouds. The goal of this work is to maximize the number of user-prioritized workflows that can be completed given budget and deadline constraints.

We developed three algorithms to solve this problem: two dynamic algorithms, DPDS and WA-DPDS, and one static algorithm, SPSS. The algorithms were evaluated via simulation on ensembles of synthetic workflows, which were generated based on statistics from real scientific applications.

The results of our simulation studies indicate that the two algorithms that take into account the structure of the workflow and task runtime estimates (WA-DPDS and SPSS) yield better results than the simple priority-based scheduling strategy (DPDS), which makes provisioning decisions based purely on resource utilization. This underscores the importance of viewing workflow ensembles as a whole rather than as individual tasks or individual workflows.

In cases where there are no provisioning delays, task runtime estimates are good, and failures are rare, we found that SPSS performs significantly better than both dynamic algorithms. However, when conditions are less than perfect, the static plans produced by SPSS are disrupted and it frequently exceeds the budget and deadline constraints. In comparison, the dynamic algorithms are able to adapt to a wide variety of conditions, and rarely exceed the constraints even with long delays, poor estimates, and high failure rates.

This study suggests several areas for future work. Our current approach models data access as part of the task execution time and does not consider data storage and transfer costs. In the future we plan to extend the application and infrastructure model to include the various data storage options available on clouds. A previous experimental study [26] suggests that the data demands of scientific workflows have a large impact on not only the execution time, but also on the cost of workflows in commercial clouds. In dynamic algorithms, it will be interesting to extend our utilization-based autoscaling with more advanced workflow-aware strategies based on feedback control. We also plan to investigate heterogeneous environments that include multiple VM types and cloud providers, including private and community clouds, which will make the problem even more complex and challenging.

Acknowledgments

This work was supported by the NSF under award OCI-0943725 (STCI) and by the EU FP7 projects UrbanFlood (248767) and VPH-Share (269978).

REFERENCES

[1] S. Callaghan, P. Maechling, P. Small, K. Milner, G. Juve, T. Jordan, E. Deelman, G. Mehta, K. Vahi, D. Gunter, K. Beattie, and C. X. Brooks, "Metrics for heterogeneous scientific workflows: A case study of an earthquake science application," *International Journal of High Performance Computing Applications*, vol. 25, no. 3, pp. 274–285, 2011.

[2] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in *2008 ACM/IEEE Conference on Supercomputing (SC 08)*, 2008.

[3] J. Vöckler, G. Juve, E. Deelman, M. Rynge, and G. B. Berriman, "Experiences using cloud computing for a scientific workflow application," in *2nd Workshop on Scientific Cloud Computing (ScienceCloud '11)*, 2011.

[4] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, O. Akan *et al.*, Eds. Springer Berlin Heidelberg, 2010, vol. 34, pp. 115–131.

[5] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes, "Sky computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43–51, 2009.

[6] D. Durkee, "Why cloud computing will never be free," *Communications of the ACM*, vol. 53, no. 5, pp. 62–69, May 2010.

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[8] "Amazon Auto Scaling." [Online]. Available: http://aws.amazon.com/autoscaling

[9] "RightScale." [Online]. Available: http://www.rightscale.com

[10] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010)*, 2010.

[11] H. Kim, Y. el-Khamra, I. Rodero, S. Jha, and M. Parashar, "Autonomic management of application workflows on hybrid computing infrastructure," *Scientific Programming*, vol. 19, pp. 75–89, April 2011.

[12] J. Yu, R. Buyya, and C. Tham, "Cost-Based scheduling of scientific workflow application on utility grids," in *First International Conference on e-Science and Grid Computing*, 2005.

[13] S. Abrishami, M. Naghibzadeh, and D. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," in *11th IEEE/ACM International Conference on Grid Computing*, 2010.

[14] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *2011 ACM/IEEE Conference on Supercomputing (SC '11)*, 2011.

[15] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, Mar. 2009.

[16] R. Prodan and M. Wieczorek, "Bi-Criteria Scheduling of Scientific Grid Workflows," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 364–376, 2010.

[17] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems," in *19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '07)*, 2007.

[18] A. K. M. K. A. Talukder, M. Kirley, and R. Buyya, "Multiobjective differential evolution for scheduling workflow applications on global Grids," *Concurrency Computation: Practice and Experience*, vol. 21, no. 13, pp. 1742–1756, 2009.

[19] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *International Conference on Advanced Information Networking and Applications*, 2010.

[20] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *Integrated Research in GRID Computing*, ser. COREGrid Series. Springer-Verlag, 2007.

[21] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling, "Scientific workflow applications on amazon EC2," in *2009 5th IEEE International Conference on E-Science Workshops*, Dec. 2009.

[22] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi, "Characterization of scientific workflows," in *3rd Workshop on Workflows in Support of Large Scale Science (WORKS 08)*, 2008.

[23] "Workflow Generator." [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

[24] "Cloud Workflow Simulator." [Online]. Available: https://github.com/malawski/cloudworkflowsimulator

[25] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[26] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. P. Berman, B. Berriman, and P. Maechling, "Data sharing options for scientific workflows on amazon EC2," in *2010 ACM/IEEE conference on Supercomputing (SC 10)*, 2010.

[27] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems," UCSB Computer Science Technical Report 2008-10, 2008.

[28] G. Juve and E. Deelman, "Automating Application Deployment in Infrastructure Clouds," in *3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, 2011.

[29] National Energy Research Scientific Computing Center (NERSC), "Magellan." [Online]. Available: http://magellan.nersc.gov

[30] "FutureGrid." [Online]. Available: http://futuregrid.org/